

Python for Data Science

5. Object Oriented Programming



Contents

5.1 Class Definition

5.2 Controlling Access to Attributes

5.3 Inheritance

1

Class Definition

Defining a Class

A class definition begins with the keyword `class` (line 5) followed by the class's name and a colon (`:`). This line is called the **class header**.

Each class typically provides a descriptive *docstring* (line 6). When provided, it must appear in the line or lines immediately following the class header.

The identifier `Account` is both the class name and the name used in a constructor expression to create an `Account` object and invoke the class's `__init__` method.

```
1  # account.py
2  """Account class definition."""
3  from decimal import Decimal
4
5  class Account:
6      """Account class for maintaining a bank account balance."""
7
```

1

Class Definition

Creating Objects

Constructor expressions create new objects and initialize their data using arguments specified in parentheses.

The parentheses following the class name are required, even if there are no arguments.

```
In [3]: account1 = Account('John Green', Decimal('50.00'))
```

Getting an Account's Name and Balance

```
In [4]: account1.name  
Out[4]: 'John Green'
```

```
In [5]: account1.balance  
Out[5]: Decimal('50.00')
```

```
In [6]: account1.deposit(Decimal('25.53'))
```

```
In [7]: account1.balance  
Out[7]: Decimal('75.53')
```

1

Class Definition

Initializing Objects

The constructor expression creates a new object, then initializes its data by calling the class's `__init__` method.

Each new class you create can provide an `__init__` method that specifies how to initialize an object's data attributes.

Returning a value other than `None` from `__init__` results in a `TypeError`.

```
8 def __init__(self, name, balance):
9     """Initialize an Account object."""
0
1     # if balance is less than 0.00, raise an exception
2     if balance < Decimal('0.00'):
3         raise ValueError('Initial balance must be >= to 0.00.')
4
5     self.name = name
6     self.balance = balance
7
```

Initializing Objects

When you call a method for a specific object, Python implicitly passes a reference to that object as the method's first argument. For this reason, all methods of a class must specify at least one parameter.

By convention most Python programmers call a method's first parameter `self`. A class's methods must use that reference (`self`) to access the object's attributes and other methods.

Class `Account`'s `__init__` method also specifies parameters for the name and balance.

When an object of class `Account` is created, it does not yet have any attributes. They're added *dynamically* via assignments of the form:

```
self.attribute_name = value
```

1

Class Definition

method

The Account class's `deposit` method adds a positive amount to the account's `balance` attribute.

If the amount argument is less than 0.00, the method raises a `ValueError`, indicating that only positive deposit amounts are allowed.

If the amount is valid, line 25 adds it to the object's `balance` attribute.

```
18 def deposit(self, amount):
19     """Deposit money to the account."""
20
21     # if amount is less than 0.00, raise an exception
22     if amount < Decimal('0.00'):
23         raise ValueError('amount must be positive.')
24
25     self.balance += amount
```

2

Controlling Access to Attributes

Encapsulation

A class's **client code** is any code that uses objects of the class.

Most object-oriented programming languages enable you to **encapsulate** (or **hide**) an object's data from the client code. Such data in these languages is said to be *private data*.

Leading Underscore Naming Convention

Python does *not* have private data. Instead, you use *naming conventions* to design classes that encourage correct use.

By convention, Python programmers know that any attribute name beginning with an underscore (_) is for a class's *internal use* only.

Attributes whose identifiers do *not* begin with an underscore (_) are considered publicly accessible for use in client code.

3

Inheritance

Base Classes and Subclasses

Often, an object of one class is *an* object of another class as well.

For example, a `CarLoan` is a `Loan`. Class `CarLoan` can be said to inherit from class `Loan`.

In this context, class `Loan` is a **base class**, and class `CarLoan` is a **subclass**. A `CarLoan` is a specific type of `Loan`, but it's incorrect to claim that every `Loan` is a `CarLoan`.

Base class	Subclasses
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
BankAccount	CheckingAccount, SavingsAccount

3

Inheritance

Inheritance Hierarchy

Inheritance relationships form tree-like *hierarchical* structures.

A base class exists in a hierarchical relationship with its subclasses.

With **single inheritance**, a class is derived from *one* base class.

With **multiple inheritance**, a subclass inherits from *two* or *more* base classes.

